A Robust Search–Based Approach to Project Management in the Presence of Abandonment, Rework, Error and Uncertainty

Giuliano Antoniol* and Massimiliano Di Penta* and Mark Harman**
antoniol@ieee.org dipenta@unisannio.it mark.harman@brunel.ac.uk

* RCOST - Research Centre on Software Technology
University of Sannio, Department of Engineering
Palazzo ex Poste, Via Traiano 82100 Benevento, Italy

**Department of Information Systems and Computing,
Brunel University, Uxbridge, Middlesex, UB8 3PH, UK

Abstract

Managing a large software project involves initial estimates that may turn out to be erroneous or that might be expressed with some degree of uncertainty. Furthermore, as the project progresses, it often becomes necessary to rework some of the work packages that make up the overall project. Other work packages might have to be abandoned for a variety of reasons. In the presence of these difficulties, optimal allocation of staff to project teams and teams to work packages is far from trivial.

This paper shows how genetic algorithms can be combined with a queuing simulation model to address these problems in a robust manner. A tandem genetic algorithm is used to search for the best sequence in which to process work packages and the best allocation of staff to project teams. The simulation model, that computes the project estimated completion date, guides the search. The possible impact of rework, abandonment and erroneous or uncertain initial estimates are characterised by separate error distributions.

The paper presents results from the application of these techniques to data obtained from a large scale commercial software maintenance project.

Keywords: Software Project Management, Genetic Algorithms, Queuing Simulation

1 Introduction

Planning a large scale software project involves a set of initial effort estimates and an allocation of programmers to teams and teams to work packages (WPs). The work packages are typically derived from a Work Breakdown Structure (WBS), with each WP being assigned to a particular team.

In a perfect world, the initial estimates would be accurate and the teams would successfully complete the WPs in the estimated time. No WP would need to be abandoned and there would be no need to revisit and re-work a WP once complete. Of course, we do not live in a perfect world and issues of erroneous, inaccurate or uncertain estimates are known to plague the manager's role in assigning staff to teams and teams to WPs.

Furthermore, even if the initial effort estimates for each WP turned out to be astonishingly accurate and certain, there are eventualities which might crop up during the progress of the project which need to be taken into account as far as possible, in the initial allocations. Two of these are the need of re-work and abandonment. Rework may be required for a variety of reasons. For instance, the specification of a WP might change, or there may be a suspicion over the way in which a WP was initially carried out. Likewise, there are many reasons why a WP might be abandoned; perhaps it turned out to be no longer required, perhaps it was simply impossible to complete or it became a casualty of forced economy.

In this paper we study this problem from the perspective of a massive software maintenance project (such as Y2K remediation, Euro conversion or phone numbering change) involving a large number of applications simultaneously [14]. Such maintenance activities present particularly acute problems for managers, since they have fixed hard deadlines and cut right across an entire software portfolio, touching almost every software asset possessed by the

Proceedings of the 10th International Symposium on Software Metrics (METRICS'04)
1530-1435/04 $ 20.00 IEEE

organisation.

Having obtained initial effort estimates, the next task is to determine the staffing level (i.e., the number of people needed, and their distribution among teams) and the project planning (i.e., the WP assignment to teams). Resource allocation is likely to be subject to constraints and negotiation. Corporate guidelines, strategies and available budget, as well as the experience on previous projects may serve to determine an early estimate of the staffing level (i.e., number of allocated people). Usually, at project inception resources are negotiable and, more generally, re-staffing may occur during the project life thus team organisation may vary over time.

Given a fixed amount of resource available, project managers have to optimize them by i) determining the distribution of people into teams and ii) by assigning maintenance activities to teams. The two factors are clearly interwoven. Given a fixed distribution of people into teams, there exists a WP assignment to teams which minimizes the time required to carry out the project. This can also be thought of as the optimal order in which the WPs flow into a queuing network that models the maintenance process [3].

On the other hand, given a fixed WP ordering, there exists an optimal people distribution into teams. Such resource allocation problems are instances of the 'bin packing problem', the solution of which is NP-hard [8] and, for which, search based techniques are known to be effective [13].

This paper proposes a tandem-approach to account for two distinct phases. In a first stage, for a given staffing level and a random or uniform people distribution across teams, an optimal WPs ordering is determined. In a second stage, given the staffing level and the WPs ordering previously determined an optimal organisation of teams is computed.

The two steps are repeated until a negligible planning modification is observed. The two different stages rely on Genetic Algorithms (GAs) to find the optimal WP ordering and the optimal staff allocation. In particular, a genome encoding, modelling the WP ordering problem, is used. The fitness function has been implemented basing on a queuing simulator described in [3]. The approach caters for the effects of re-work and abandonment and the uncertainty inherent in the effort estimation process [24].

The approach is validated by an empirical study, using historical data from a real–world massive Y2K maintenance intervention, conducted on a financial system for a European company. Sensitivity analysis is used to evaluate the effect of manager uncertainty in the effort estimation, as well as the effect of rework on WP or abandonment (i.e., cases in which some stages of a maintenance task are not needed).

The primary contributions of this paper are as follows:

- The paper considers the problem of software project resource allocation and project planning using Search–Based approaches. In particular, a tandem–approach, aiming at determining two different factors of the project, i.e., the people distribution and WP assignment to teams, is proposed;
- The paper explicitly accommodates into the allocation process both abandonment and rework allowing for quantification of the effect on the overall project planning for different hypothesized rework and abandon levels;
- The paper also presents results of a study into the effect of project manager error in estimating the effort required to accomplish WPs maintenance.

The remainder of the paper is organised as follows. After a brief overview of existing scheduling approaches and application of heuristic approaches to software project management, Section 3 presents, for sake of completeness, an introduction to GA and stochastic simulation, while Section 4 describes the proposed approach. Section 5 reports and discusses the results from the empirical study, also indicating which are the assumptions made. Section 6 concludes.

2 Related Work

An application of search–based techniques to project scheduling was done by Davis [6]. A survey of the application of GAs to solve scheduling problems has been presented by Hart et al. in [12]. The mathematical problem encountered is, as described by the author, the classical, NP-hard, bin packing or shop-bag problem. A survey of approximated approaches for the bin packing problem is presented in [5]. More recently Falkenauer published a book devoted to the GA and grouping problems [7]. The theme of the book and the proposed genome encoding are highly relevant to the problem addressed in this paper. Schema interpretation and bin packing genome encoding described in the book were a source of inspiration although our problem is slightly different. The order on which WPs are presented to the teams is relevant whereas bin packing doesn't impose a packing order to reach an optimum.

Search heuristics have been applied in the past to solve some related software project management problems. In particular, Kirsopp et al. reported a comparison of random search, hill climbing and forward sequential selection to select the optimal set of project attributes to use in a search–based approach to estimating project effort [17]. A comparison of approaches (both analytical and evolutionary) for prioritizing software requirements is proposed in [15], while Greer and Ruhe proposed a GA-based approach for planning software releases [10].

More commonalities can be found with the work of Antoniol et al. [3]. The paper [3] focuses on problem of staffing a software maintenance project using queuing networks and discrete-event simulation. Given an (ordered) distribution of incoming maintenance requests, the goal of Antoniol et al. was to determine the staffing levels for each team. The present paper relies on that approach to determine the initial staffing, and on the queuing simulator proposed to implement the objective function of our GAs.

Issues connected to empirical studies of software process simulation modelling are also discussed by Raffo et al. [22], with particular reference to the estimate of simulation parameters from real-world data, and to compare actual results to the model's results. Abdel-Hamid [2] published an approach based on system dynamics modelling to verify the degree of interchangeability of men and months [4]. Results from the analysed case study do not fully support Brooks' law.

Queueing theory was also recently applied by Ramaswamy [23] to model software maintenance requests. Simulations of a software maintenance process were performed by Podnar and Mikac in [20] with the purpose of evaluating different process strategies rather than staffing the system.

3 Background

To tackle the project planning and resource allocation problems GA and stochastic simulation were applied to information obtained from manager estimates and project WBS.

3.1 Genetic Algorithms

GAs originate with an idea, born over 30 years ago, of applying the biological principle of evolution to artificial systems. Roughly speaking, a GA may be defined as an iterative procedure that searches for the best solution of a given problem among a constant-size population, represented by a finite string of symbols, the *genome*. The search is made starting from an initial population of individuals, often randomly generated. At each evolutionary step, individuals are evaluated using a *fitness function*. High-fitness individuals will have the highest probability to reproduce.

The evolution (i.e., the generation of a new population) is made by means of two operators: the *crossover operator* and the *mutation operator*. The crossover operator takes two individuals (the *parents*) of the old generation and exchanges parts of their genomes, producing one or more new individuals (the *offspring*). The mutation operator has been introduced to prevent convergence to local optima, in that it randomly modifies an individual's genome (e.g., by flipping some of its bits, if the genome is represented by a bit string). Crossover and mutation are performed on each individual of the population with probability *pcross* and *pmut* respectively, where $pmut \ll pcross$. Further details on GA can be found in [7, 9].

GA are effective in finding optimal (or near optimal) solutions for problems where:

- The search space is large or complex;
- No mathematical analysis is available;
- Traditional search methods did not work; and, as in our case
- The solution of the problem is NP-hard [8].

A GA does not guarantee to converge: the termination condition is often specified as a maximal number of generations, or as a given value of the fitness function.

3.2 Queuing Theory and Stochastic Simulation

A queuing system can be described as customers arriving for service, waiting for service if it is not immediate, and leaving the system after being served (by servers). The term *customer* is used in a general sense and does not necessarily imply a human customer (e.g., maintenance requests can be thought of as customers). Further details can be found in [11]. Traditional queuing theory only models the steady state of the process. In particular, it models parameters such as the waiting time of a request, or the coefficient of use of a server, given an arrival traffic rate, a service time distribution, a queue discipline and capacity. To deal with transient situations, stochastic simulation is needed.

Simulation attempts to build a model that will mimic a real system for most of its relevant aspects. A simulation may be *deterministic* or *stochastic*. In the first case, the behaviour is "point-wise" defined by the output of the model, and results are repeatable. Stochastic simulation involves randomness: multiple runs may generate different values. Moreover, a simulation may be *static* or *dynamic*, depending upon whether or not it involves time.

Finally, simulation may be further classified as *system dynamics simulation*, *discrete-event simulation* [16, 19] and *state-based simulation* [21]. In particular, discrete-event simulation describes process steps, and is well suited to model queuing systems such as system modelling software development or maintenance were requirements, use cases, test incident reports, bug reports or maintenance interventions are queued and handled by teams of programmers.

To compute the fitness function of our GAs, we modelled the software maintenance process as a queuing system. As proposed in [3] our fitness function is computed via stochastic, dynamic, discrete-event simulations.

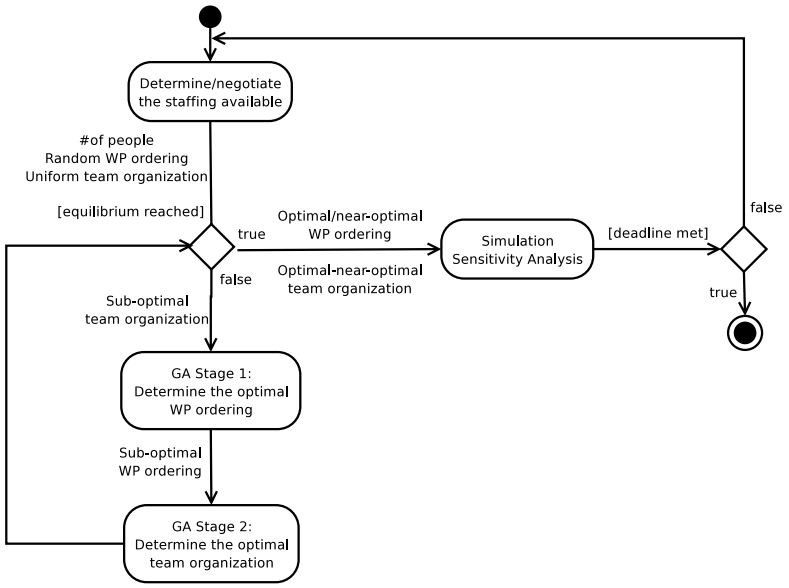

Figure 1. Activity diagram of the proposed approach

4 The Approach

This section explains how we formulated the problem stated in the introduction as a search–based problem, using a tandem, two–phased approach applying different GAs in the two phases. The approach is represented, as an activity diagram, in Figure 1.

At a first level of approximation the maintenance task is considered as a monolithic step task (i.e., the single queue, multi-server model of Antoniol et al. [3]). More sophisticated maintenance processes have no impact on the proposed approach, because the queuing model, as it will become clear later, is encapsulated by the fitness function.

The first activity of the process aims at defining the list of WPs to be maintained, and to negotiate the number of people available. Initially, people are equally distributed among the maintenance teams (i.e., the queuing model servers). In our case study, we initially instantiated singleton teams of people.

Two GAs were used in tandem. The first GA aims to find an optimal (or near optimal) solution to the problem of determining the ideal order in which to present work packages to the process; the WP ordering. The second GA aims to find a good (hopefully optimal) allocation of staff to project teams to maximize the throughput for the WP ordering found by the first GA. The overall optimisation process is iterative; the results of the second GA are fed back into the first GA, which attempts to find tune the optimisation of WP ordering[1]. This repetition continues until the search stabilises on a solution.

The initial team organisation and a random WP ordering are the seeds to the first GA, that determines, for fixed team organisation, the optimal WP ordering. The aim here is to minimize the project completion time. The obtained order is seed into the second GA, which, while maintaining the WP ordering invariant, computes the optimal grouping of staff into teams.

This newly defined team allocation is fed back into the first GA. This is necessary because changing the structure of teams may require a different WP ordering to that initially found by the first GA. The process is iterated until no substantial modification in the project planning is observed; stability has been reached (helpfully on a highly optimal solution).

Finally, the obtained ordering and staff allocation are presented to a queuing simulator that analyses the sensitivity of the result with respect to possible effort estimation errors, of potential reworks on a given percentage of maintenance tasks. Perhaps the sensitivity analysis will reveal that the solution obtained by the tandem-GA is too vulnerable to fluctuations in estimates of WP effort (perhaps arising either from error or uncertainty on the part of the manager). Furthermore, even where the solution obtained is highly robust, the ultimate project completion date might miss the projected deadline. The purpose of this final stage is to recognize these two possible problems. In either case, we will require a negotiation of further people and a successive iteration of the entire process.

[1]The process can also work by first determining the people distribution, and then the WP ordering

The next two subsections will present details of the two GAs, describing the genome encoding, the fitness function as well as the crossover and mutation operators.

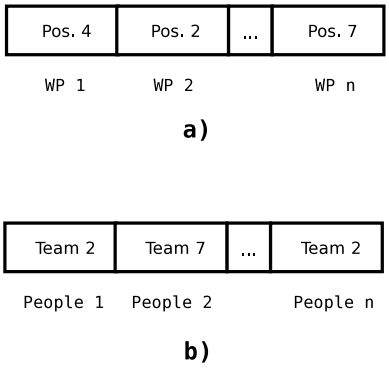

Figure 2. GA Genomes: a) WP ordering and b) People assignment to teams

4.1 GA stage 1: determining the optimal WP ordering

To encode the WP assignment to teams in a genome, we adopted a scanning genome. The genome is encoded as an N-sized array, the value of a genome element indicates the position of the WP in the incoming queue, for a single-queue/multi-server queuing system. The genome schema is shown in Figure 2-a.

The fitness function takes as input the genome (i.e., the WP sequence) and estimates the project completion time using the queuing simulator described in the paper [3]. In particular, the single-queue/multi-server model is instantiated considering the people distribution obtained at the previous iteration (or an uniform people distribution across teams for the first iteration). Once the queuing system configuration and the WP effort are known, the queuing simulator estimates the completion time simulating WP flowing in the maintenance process as time passes.

The mutation operator randomly selects two WPs (i.e., two array items) and exchanges their position in the queue. The crossover operator is somewhat more complex. Two offspring (o_1 and o_2) are formed from two parents (p_1 and p_2), as follows:

1. A random position k, is selected in the genome.
2. The first k elements of p_1 become the first k elements of o_1.
3. The last N-k elements of o_1 are the sequence of N-k elements which remain when the k elements selected from p_1 are removed from p_2.
4. o_2 is obtained similarly, composed of the first N-k elements of p_2 and the remaining elements of p_1 (when the first N-k elements of p_2 are removed).

For instance, if $k = 2$ and $p_1 \equiv \{4, 2, 3, 6\}$ and $p_2 \equiv \{4, 6, 3, 2\}$, then $o_1 \equiv \{4, 2, 6, 3\}$ and $o_2 \equiv \{4, 6, 2, 3\}$.

This approach to crossover has the advantage that it guarantees that each offspring contains precisely one position in the sequence per WP. It therefore avoids the need for repair, or some other mechanism which might be required to deal with duplication of sequence numbers in a more simple-minded crossover operator.

4.2 GA stage 2: determining the optimal assignment of people across teams

The problem to assign people to teams is an instance of the pigeon hole problem; the genome is an array of N integers, where N is the number of programmers. Each value of the array indicates the team that a programmer is assigned to. The genome schema is shown in Figure 2-b.

The fitness function is the same as that used at the previous stage (that is, the overall project duration is to be minimized). To implement a GA for this encoding, the mutation operator randomly selects a WP and randomly changes its team. It is worth pointing out that the random number generated by the mutation operator is an integer ranging from 1 to the maximum number of people available. The cardinality of the set of numbers present in the genome determines the number of teams (servers) of the queuing model. Clearly, a cardinality equal to the number of people available indicates that each team is composed of a single person, while a cardinality equal to one indicates that a single server (composed of all people) is used.

The crossover operator is the standard *single point* crossover [9].

4.3 Rework, abandonment and effort estimation error and uncertainty

After the tandem-GA has been used to determine the optimal staff distribution across teams, and the optimal WP ordering, a sensitivity analysis is performed to model the effect of i) uncertainty in the effort estimation, ii) abandonment, and iii) rework.

The queuing system used was simple in order to allow these factors to be explored in isolation; a single-queue/multi-server maintenance process was used to simulate the project activities. In such a model, rework is accounted for by an additional effort value assigned to a given percentage of WPs. The value of this additional effort is modelled by an exponential random distribution or, if fine–grain data is available (as in our case study), by repeating some activities for a random percentage or WPs (e.g.,

repeating enactment and testing, mimicking situations in which the test fails).

Abandonment occurs when some substantial maintenance phase was not needed, or became impractical. In this situation a substantial effort (or, if data is available, the effort related to some maintenance phases) might be subtracted from a given percentage of WPs. Overall, abandonment can be modelled similarly to rework.

Finally a maintenance effort estimate may be over–optimistic, over–pessimistic or neutral. This uncertainty (or error term) was modelled via a Gaussian distribution with a mean corresponding to:

- the WP actual effort (*neutral estimate*, that models a manager that both performs optimistic and pessimistic estimates),
- the WP actual effort, plus a given percentage (*over–pessimistic estimate*), or
- the WP actual effort, minus a given percentage (*over–optimistic estimate*),

and with a standard deviation corresponding to a given percentage (hereby referred as "uncertainty level") of the actual effort value.

5 Empirical Study

The empirical study proposed in this paper aims at defining a near optimal project staffing and scheduling for maintenance activities of WPs coming from a massive maintenance project, related to fixing the Y2K problem in a large financial software system from a European financial organisation.

5.1 Context description

According to its WBS, the application was decomposed into WPs, i.e. loosely coupled, elementary units (from one to nine for each application) subject to maintenance activities; each WP was managed by a WP leader and assigned to a maintenance team. Overall, the entire system was decomposed in 84 WPs, each one composed, on average, of 300 COBOL and JCL files. The project followed a phased maintenance process (similar to that defined by the IEEE maintenance standard [1]), encompassing five phases:

1. *Inventory*: deals with the decomposition of the application portfolio into WPs;
2. *Assessment*: identifies, for each WP, candidate impacted items for the maintenance task;
3. *Technical Analysis (TA)*: deals with the analysis of impacted items and identifies a candidate solution;
4. *Enactment (Enact)*: an automatic tool was used to apply Y2K patches based on windowing [18]; and
5. *Unit Testing (UT)* performed on each impacted WP.

Further details can be found in [3].

Figure 3 depicts the WP effort histogram (efforts are expressed in person days).

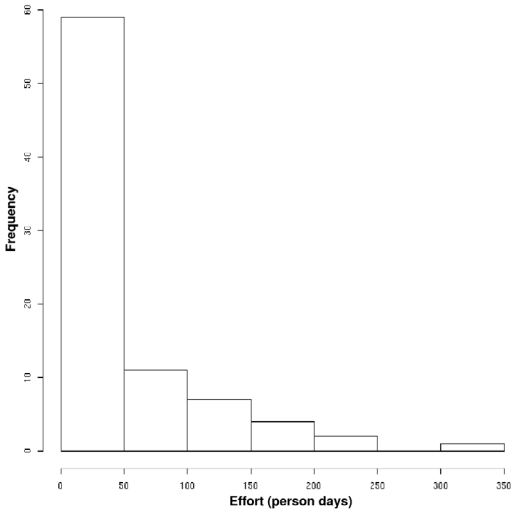

Figure 3. WP effort histogram

5.2 Assumptions

Because the maintenance intervention studied here was performed almost semi-automatically and involved highly standardized activities, it is possible to make the assumption of interchangeability between people and months. That is, given a maintenance team size, s and the effort required e, the time t necessary to perform the task is:

$$t \simeq \frac{e}{n} \tag{1}$$

Due to Brooks' law [4], this could be an overly optimistic assumption. However, as other authors have noted [2], given the small team sizes (fewer than eight people) and the standard (training–free) nature of the maintenance task, this approximation was considered reasonable. The model can be generalised to situations in which Brooks' law does apply by the simple introduction of a non-linearity factor.

A further simplification introduced with our study is the absence of dependencies between WPs (i.e., there is no constraint on possible orderings which can be selected), however, if needed such a constraint could be incorporated into the first GA, for example adopting the approach proposed in [7] to model the line balancing problem.

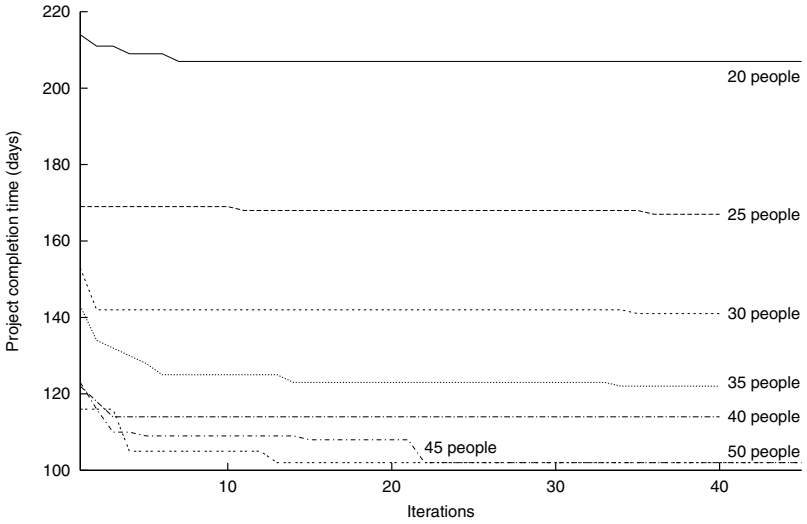

Figure 4. Estimated project completion time obtained with different number of people and different algorithm iterations

5.3 Threats to Validity

Construct validity threats may be due to the simplifications made on the maintenance process modelled, as well as to the assumptions made in Section 5.2. However, the effect of these threats is limited because i) the applicability of the GA tandem model does not depend from the topology of the maintenance process (i.e., a more complex model simply requires that the objective function should be computed over a queuing network instead of over a single-queue system) and ii) for the case study adopted there were no dependencies at all.

Internal validity threats, in our case study, can be due to the randomness of the results obtained from simulation and GA. To avoid such a threat, different actions were taken:

- First and foremost, we carefully calibrated the number of generations and population size needed by GA. The chosen values were determined ensuring that further increases do not significantly affect the results;
- Similarly, we calibrated the number of iterations required by the whole approach. As it will be detailed in Section 5.4, further increases over 40 iterations do not produce improvements in our fitness function.
- To avoid results being affected by randomness, GAs were executed 20 times and we verified that the obtained fitness function at the last stage do not change among the iterations.

With regards to external validity, as explained in Section 5.2 our approach and the results obtained can be extended *as is* to situations in which i) the Brooks' law is not valid and ii) there are no dependencies between WPs. For all the other cases, there is the need for a more complex model. First and foremost, there may be the need to account for non-linearity factors in Equation (1), or to consider the constraints due to the WP dependencies.

Finally, as detailed in Section 4.3 and as it will be shown in Section 5.4.2, realistic situations such as rework, abandonment and effort estimate uncertainty have been accounted in the overall approach.

5.4 Empirical study results

This section reports results obtained re-planning the above described case study with the proposed approach.

Simulations were run considering the following parameters:

- elitist GA;
- 100 individuals;
- 1000 generation for each GA;
- mutation probability 0.1, crossover probability 0.6;
- 40 iterations of the tandem (we experienced no further improvement after that, until 100 iterations).

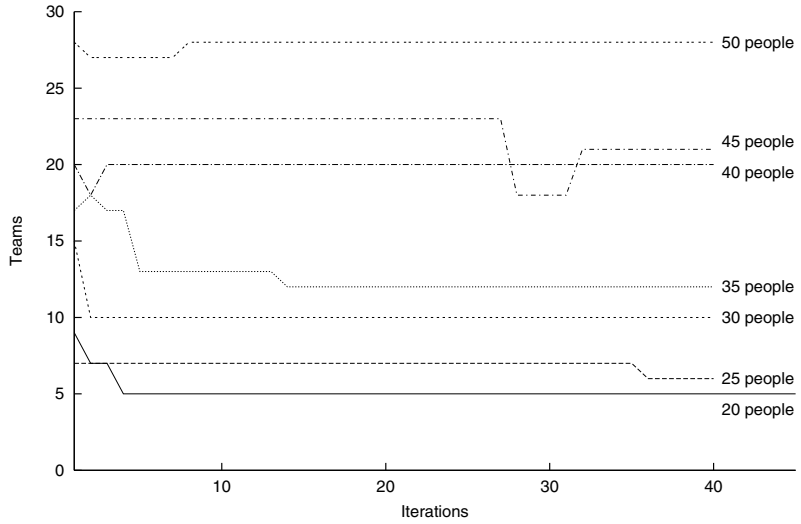

Figure 5. Number of teams (servers) allocated with different number of people and different algorithm iterations

5.4.1 Determining the WP ordering and team allocation: the ideal case

Figure 4 shows how the tandem approach adjusts the WP ordering and the team organisation with the aim of minimizing the number of days required for the maintenance task, hereby referred as "project completion time". It is worth pointing out how, for both staffing levels of 45 and 50 people, the completion time obtained is the same (although the algorithm is slightly slower to converge for 45 people). This indicates an "upper-bound" in the staffing level, over which there is no substantial improvement in terms of time required. Not surprisingly, 45 people corresponds to the optimal staffing level indicated in [3].

In correspondence to each minimum project completion time determined for each staffing level, the algorithm outputs the optimal WP ordering and the people distribution across teams. Figure 5 plots the optimal number of teams allocated, for each staffing level, after a given number of iterations of the algorithm. It is worth pointing out how sometimes, in correspondence to the optimal critical path, the number of teams is subject to oscillations (as the cases for 45 and 50 people). In other words, the algorithm tries to reduce the number of teams but, after exploring the search space corresponding to the new number of teams, is not able to improve the fitness function anymore, thus it restores the previous value.

For each staffing level (i.e., total number of people available) and number of teams, the algorithm determines a a distribution of people across teams. Table 1 reports details about such a distribution, indicating, for each staffing level, the number of teams of each size the algorithm has allocated. For example, given a staffing level of 20 people, the algorithm has allocated two teams of 5 people, two of four people and one of two people.

Team Size	Staffing Level						
	20	**25**	**30**	**35**	**40**	**45**	**50**
1	-	-	2	1	7	5	10
2	1	1	1	3	7	10	14
3	-	1	4	5	5	4	4
4	2	1	1	2	1	2	-
5	2	2	2	1	-	-	-
6	-	1	-	-	-	-	-

Table 1. Number of teams of different sizes for different staffing levels

The table gives us several insights about how the algorithm tends to allocate people for different staffing levels, and which could be the risk this implies.

When few people are available, the algorithm tends to create large teams. The phenomenon is highlighted by the 5 people teams created with staffing levels of 20, 25 and 30 people, as well as the 6 people team created with a staffing level of 25 people. This can be explained by the fact that, when the number of people available is not very

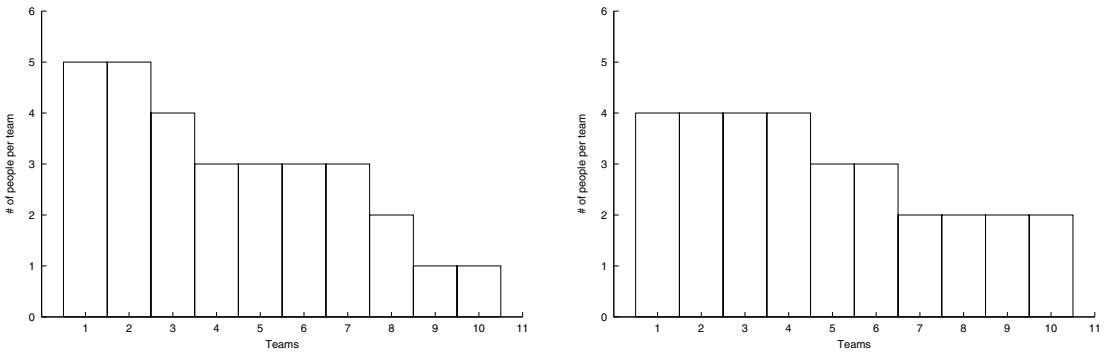

Figure 6. People allocation across teams with a staffing of 30 people a) no threshold b) minimum 2 people per team

high, they should work together to quickly accomplish the longest tasks (WP). Each incoming WP that requires a big effort is thus dispatched to such a team. Since the staffing level of that team is high, the task is completed very quickly, and the team will be ready for any next incoming long task. On the contrary, if the staffing level is higher, it is preferable to have more medium/small size teams. Such teams will take more time to accomplish a long task; however, each other incoming (long) task will not have to wait in the queue, since it can be dispatched to one of the other teams.

Attention should however be paid to avoid creating excessively large teams. Although for our case study (see Section 5.2) the communication overhead is almost negligible and Brooks' law not applicable, there is still an upper bound over which further increase of a team size could produce a negative effect. This parameter is to be decided case by case from the project manager, and can easily be modelled in our GA by introducing a penalty factor in the fitness function of the second stage (i.e., the stage that determines the people allocation) when the maximum team size exceeds a given threshold.

On the other hand, when the number of available people increases, the algorithm tends to create a larger number of small teams, even singleton teams. From one point of view, this tends to parallelize small tasks (that constitutes, as highlighted form Figure 3, the largest part of our case study). However, having single people teams can be cause of unexpected delays, since the task outcome totally depends from the availability and performance level of the single person. The project manager should avoid that whenever possible, by i) considering, as a working unit, pairs instead singletons; or ii) similarly to the previous case, relying on a GA penalty factor for teams below a given threshold.

Following the second approach, we run again the tandem GA for the 30 person staffing level, introducing a penalty for teams below 2 people. Figure 6 plots the size of each team determined with (Figure 6–b) or without (Figure 6–a) the penalty factor. The figure shows how the maximum team size has been reduced (from 5 to 4), and people more uniformly distributed across teams, avoiding singletons. Interestingly, the achieved project completion time was exactly the same.

People	Average	Min	Max	Std. Dev.
20	201.25	198	204	2.38
25	164.83	163	167	1.46
30	139.14	133	141	2.53
35	119.50	117	122	1.44
40	106.10	99	114	5.00
45	93.95	82	102	6.03
50	82.29	65	102	13.25

Table 2. Project completion time variability across teams

5.4.2 The real world model: accounting for abandonment, rework and uncertainty

For a given people distribution and estimated project completion time, it is worth knowing the resource usage during the maintenance period. In other words, the estimated project completion time of each team gives us information about the availability of people at a given date. When some teams finish their task, the manager could use resource for i) back-up to cover for emergencies in other teams. Of course here, the manager would need to pay attention to avoid the mythical man month problem [4]; or ii) allocating free resources on other projects. In addition, the variability

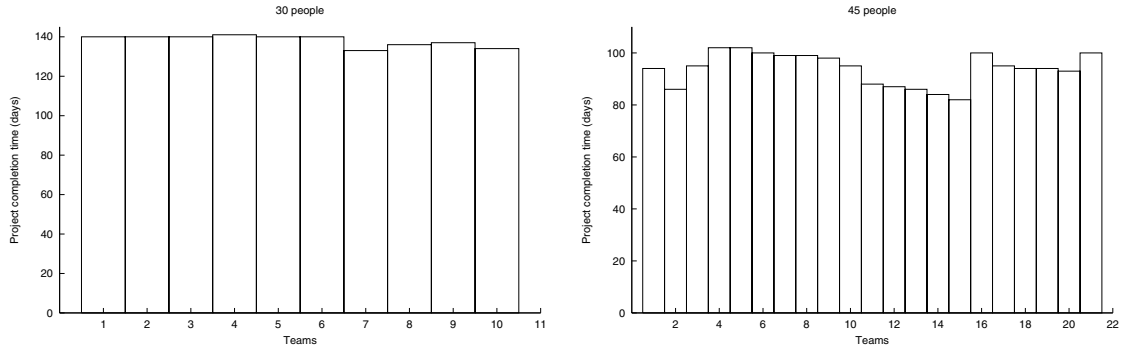

Figure 7. Examples of resulting Gantt charts

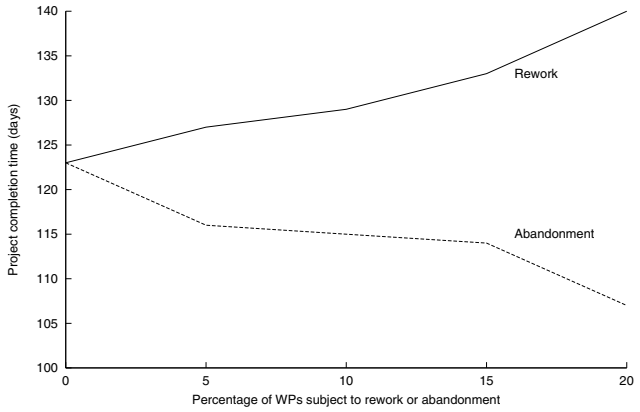

Figure 8. Project completion time variation in case of abandonment and rework (for a staffing level of 35 people)

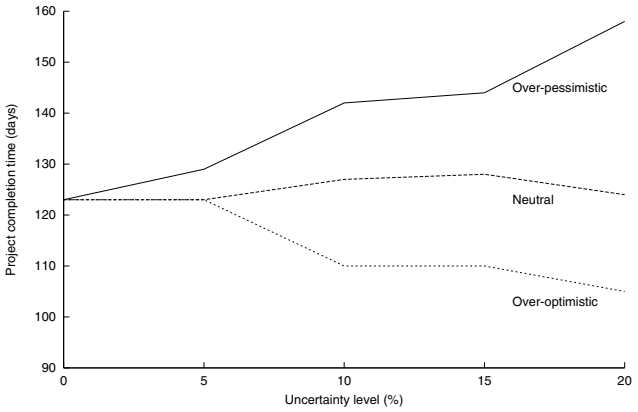

Figure 9. Project completion time variation in case of effort estimate uncertainty (for a staffing level of 35 people)

across teams also indicates how optimal is, given an available staffing level, the resource usage the algorithm is able to achieve.

As shown in Table 2, while for small numbers of people available the resource usage is optimal (i.e., the project completion time variability across teams is limited), the variability tends to increase when distributing a larger staffing (i.e., from 40 to 50 people). Summary results shown in the table are confirmed by the detailed Gantt shown in Figure 7: while for 30 people all the teams finish around 140 days, when the number of available people increases (e.g., 45 people), the algorithm is not able anymore to determine a people allocation that, at the same time, both minimizes the project completion time and maximizes the resource usage.

A possible explanation of this result could be found in the fitness function we chose (that minimizes, among all teams, the maximum project completion time, regardless of what happens for all the other teams). However, we also tested a different fitness function, aiming at minimizing the project completion time standard deviation, obtaining the same completion time and variability.

The proposed approach implies that the manager is able to obtain precise effort estimate. This, however, is not realistic: estimate errors should be modelled and simulated. Moreover, as detailed in [3], it may happen that some maintenance phases require a rework. For example, testing may fail and thus require further implementation and testing activities. On the other hand, there may be some WPs that do

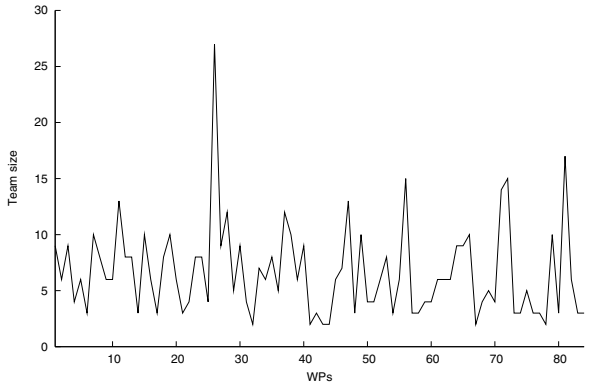

Figure 10. Sizes of the teams working on the projects

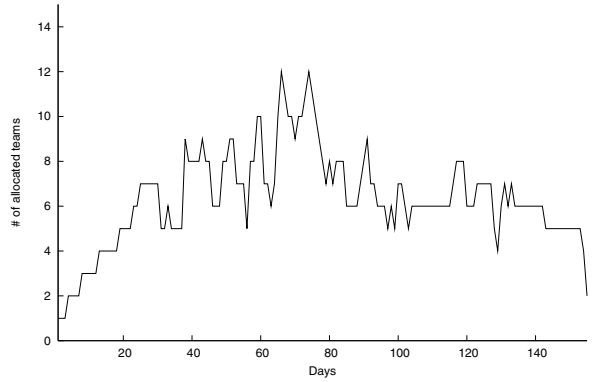

Figure 11. Number of teams staffed in the actual project at a given day

not require some phases, e.g. we may realize, after technical analysis, that no enactment is needed.

To analyze the impact of rework and abandonment, we randomly selected a percentage of the WPs (from 5% to 20%) and simulated rework on enactment and testing, as well as abandonment after technical analysis (i.e., no enactment and testing). Figure 8 show, for a staffing level of 35 people, how the project completion time changes with abandonment and rework. These curves allow the project manager to better simulate realistic situations and, if needed, re-calibrate the staffing and/or delay the project deadline. Being the abandonment and rework modelled in the same way, their effect is symmetric.

Finally, Figure 9 shows which is the variation of the completion time with the uncertainty in the effort estimate. For the *neutral estimate* the figure shows that, as the uncertainty level increases, there is an increment of the estimated completion time, due to the fact that the instantiated project planning (i.e., the above obtained WP ordering and people allocation) suffers from such effort variations. However, with further increases of the uncertainty level, the positive variability (due to pessimistic estimates) is balanced by the negative variability (due to optimistic estimates) and thus the project planning appears to be robust against such variations.

The other two curves models the *over–optimistic* and *over–pessimistic* estimates, highlighting how risky could be the former. The *over–pessimistic* estimate can cause a project over–staffing. However, it is rather conservative, allowing the managers to account for real-world project risks.

5.4.3 Actual project staffing

Finally, it is interesting to compare the results of our approach with the real project staffing level and people organisation. The staffing level was of about 80 people, however not working full-time on this project, and the completion time of 155 working days. The number of teams and the people allocation among teams varied during the time, as shown in [3]. Figure 10 reports the size of the teams that worked on each WP. Such a size varies from 2 to 27, with a median value of 6.

The number of teams working concurrently in a given day (thus the number of servants in a queuing model) is shown in Figure 11. Such a number varies from 1 to 12, with a median value of 6. This confirms the fact that industry managers avoided having a large number of teams composed of few people, preferring, instead, few teams composed of more people. This permitted to minimize the risks, also because people were also working on other, different tasks. For example, a deadline on a concurrent project could have seriously affected the project success if over-staffing were not performed.

6 Conclusion

This paper presented an approach to determine the optimal people allocation across teams and the optimal WP assignment to teams, with the aim to minimize the project completion time. The approach is based on a tandem of GAs, the first aiming at determining the WP assignment, the second aiming at determining the people allocation. Queuing simulation has been used to implement the fitness function.

The results obtained on the available data showed that, over 45 people, there is no improvement in terms of completion time, basically confirming the results obtained in [3]. The results also showed that, while with few people available the algorithm tends to create few large teams, increasing the staffing level several medium/small teams are preferred. In both case, the managers should carefully determine the upper and lower bounds, to avoid communication overhead in large groups or, even worse, singleton groups, for which the single people availability or performance can seriously affect the project likelihood of success.

With a low staffing level, the algorithm also tends to optimize the resource usage, as highlighted by the small variability across the completion time of different teams. As the staffing level increases, an optimal resource usage can result as difficult, even choosing alternative fitness functions, such as minimizing the standard deviation across teams.

Finally, we simulated the effect of rework, abandonment and uncertainty estimate, to model realistic maintenance tasks as well as manager errors, thus making our approach applicable in the practice. Clearly, tool suggestion should serve as a support to manager's decisions, also influenced, of course, by factors such as people domain expertise, availability and seniority.

Work-in-progress is devoted to apply the approach on further case study, to improve the external validity and, above all, to apply it in situations in which there are dependency relationships among WPs. A comparison with other optimization techniques, such as constraint programming, will also be performed.

References

[1] *IEEE std 1219: Standard for Software maintenance*. 1998.

[2] T. Abdel-Hamid. The dynamics of software project staffing: a system dynamics based simulation approach. *IEEE Transactions on Software Engineering*, 15(2):109–119, 1989.

[3] G. Antoniol, A. Cimitile, G. A. Di Lucca, and M. Di Penta. Assessing staffing needs for a software maintenance project through queuing simulation. *IEEE Transactions on Software Engineering*, 30(1):43–58, Jan 2004.

[4] F. Brooks. *The Mythical Man-Month 20th anniversary edition*. Addison-Wesley Publishing Company, Reading, MA, 1995.

[5] E. J. Coffman, M. Garey, and D. Johnson. Approximation algorithms for bin-packing. *In Algorithm Design for Computer System Design*, 1984.

[6] L. Davis. Job-shop scheduling with genetic algorithms. In *International Conference on GAs*, pages 136–140. Lawrence Erlbaum, 1985.

[7] E. Falkenauer. *Genetic Algorithms and Grouping Problems*. Wiley-Inter Science, Wiley - NY, 1998.

[8] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub Co, Jan 1989.

[10] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, (to appear).

[11] D. Gross and C. Harris. *Foundamentals of Queueing Theory*. John Wiley & Sons, New York, NY 10158-0012, 1998.

[12] E. Hart, D. Corne, and P. Ross. The state of the art in evolutionary scheduling. *Genetic Programming and Evolvable Machines*, 2004 (to appear).

[13] P. Husbands. Genetic algorithms for scheduling.

[14] C. Jones. Mass-updates and software project management in is organizations- *http://www.artemis.it/artemis/artemis/lang_en/libreria/mass-updates.htm*, 1999. Accessed on Aug, 25 2003.

[15] J. Karlsson, C. Wohlin, and B. Regnell. An evaluation of methods for priorizing software requirements. *Information and Software Technology*, 39:939–947, 1998.

[16] M. I. Kellner, R. J. Madachy, and D. M. Raffo. Software process simulation modelling: Why? what? how? *Journal of Systems and Software*, 46(2/3):91–106, 1999.

[17] C. Kirsopp, M. Sheppard, and J. Hart. Search heuristics, case-based reasoning and software project effort prediction. In *Proceedings of Genetic and Evolutionary Computation Conference*. Springer-Verlag, 2002.

[18] E. Lynd. Living with the 2-digit year, year 2000 maintenance using a procedural solution. In *Proceedings of IEEE International Conference on Software Maintenance*, pages 206–212, Bari, Italy, 1997.

[19] R. H. Martin and D. M. Raffo. A model of the software development process using both continuous and discrete models. *International Journal of Software Process Improvement and Practice*, 5(2/3):147–157, 2000.

[20] I. Podnar and B. Mikac. Software maintenance process analysis using discrete-event simulation. In *European Conference on Software Maintenance and Reengineering*, pages 192–195, Lisbon, Portugal, March 2001. IEEE Society Press.

[21] D. M. Raffo, W. Harrison, and J. Vandeville. Coordinating models and metrics to manage software projects. *International Journal of Software Process Improvement and Practice*, 5(2/3):159–168, 2000.

[22] D. M. Raffo and M. Kellner. Empirical analysis in software process simulation modeling. *Journal of Systems and Software*, 47(9), 2000.

[23] R. Ramaswamy. How to staff business critical maintenance projects. *IEEE Software*, 7(7):90–95, May 2000.

[24] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(11):736–743, 1997.